“High-Integrity Software, Computation and the Scientific Method"

Les Hatton

Professor of Forensic Software Engineering
CISM, Kingston University
L.Hatton@kingston.ac.uk

Version 1.1: 29/Sep/2011

High-Integrity Software

- High-Integrity Software and …
  - The scientific method
  - Defect
  - Programming languages
  - Process
  - FAQs and ways forward
- Conclusions

… the Scientific Method

- Popperian deniability …
  - Truth cannot be verified by scientific testing, it can only be falsified.
  - Falsification requires quantification of experimental error.
  - This has been at the heart of scientific progress.
  - This process is NOT generally followed in scientific (or indeed any other kind of) computation.

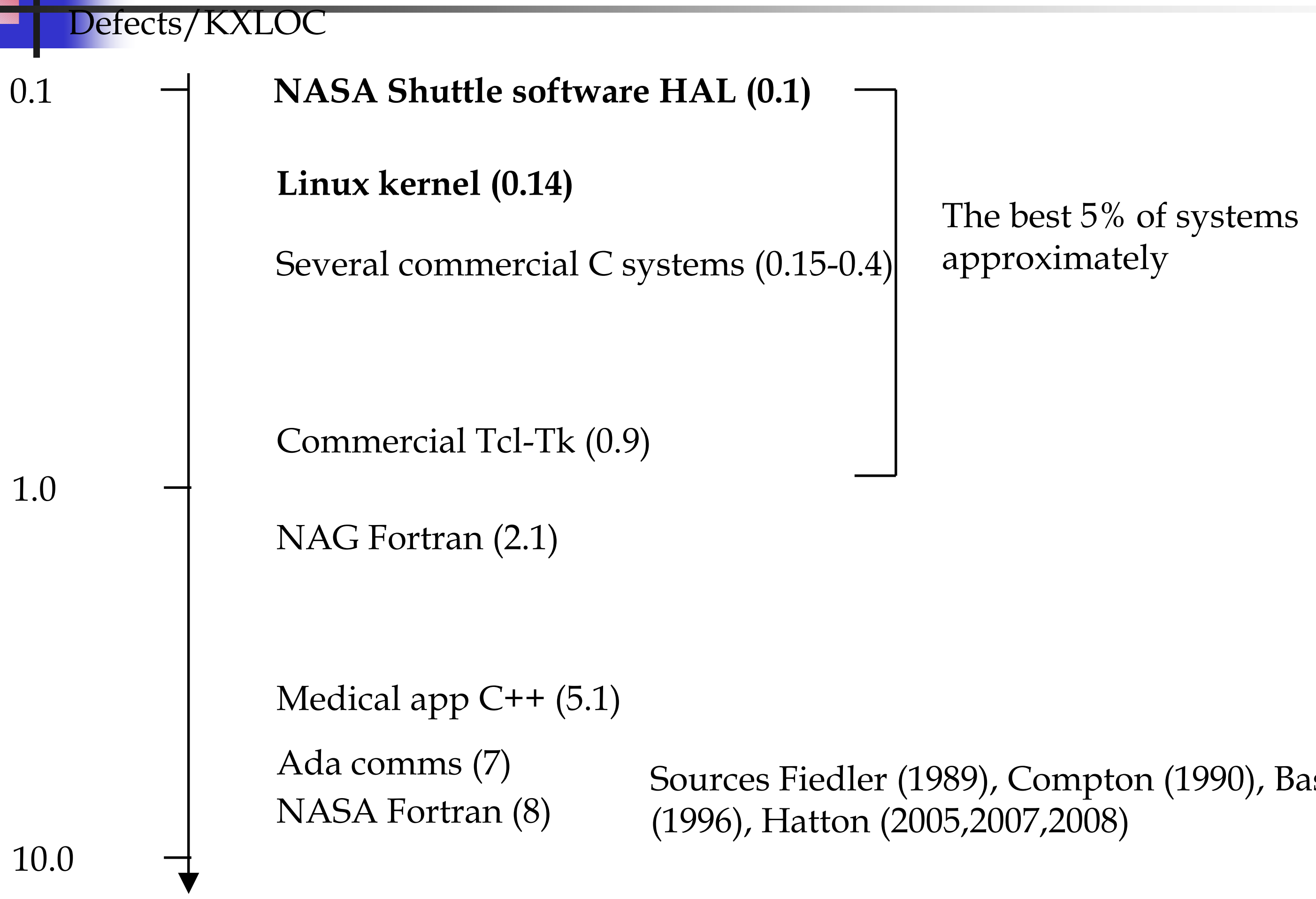
… defect

- On quantification
  - Computer scientists have researched the average *density* of defect in code extensively
  - Where we have been much less successful is in quantifying the *effects* of such defect on numerical results.

… defect

- On quantification of density
  - A “low defect” piece of software will exhibit less than 1 defect per thousand executable lines of source code in its entire lifetime.
  - Average software is in the range 1-10.

… defect.
A software quality scale based on defect density



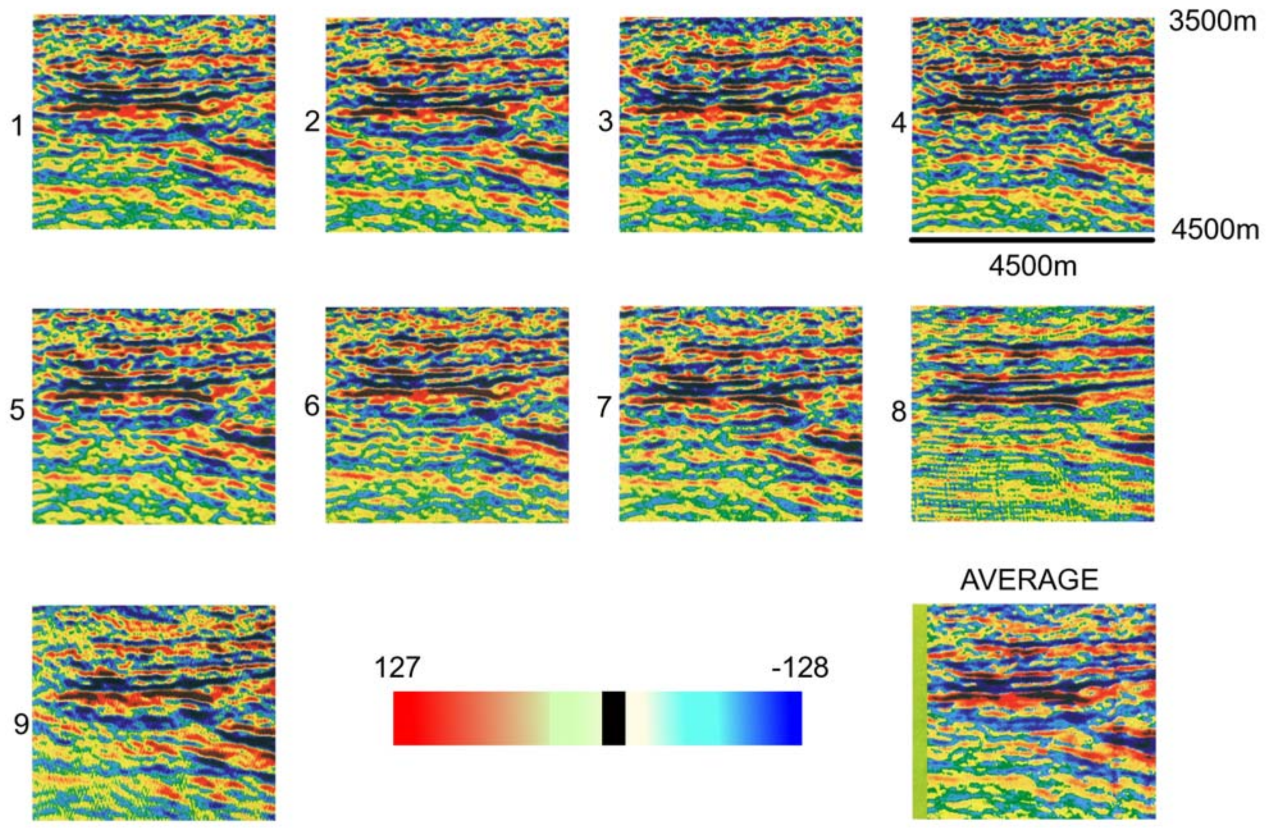
Sources Fiedler (1989), Compton (1990), Basili (1996), Hatton (2005,2007,2008)

… defect
some early thoughts

On quantification of the *effect* of defect, by 2010 I was reasonably convinced that:

- N-version experiments are exceedingly valuable at highlighting differences, (for whatever reason), and effective at reducing those differences. (1994)
- Scientific software is littered with statically detectable faults which fail with a certain frequency (1997)
- The language does not seem to make much difference. (1999-)
- Defects appear to be fundamentally statistical rather than predictive, (2005-8)
- Software systems exhibit implementation INdependent behaviour (2007-10).

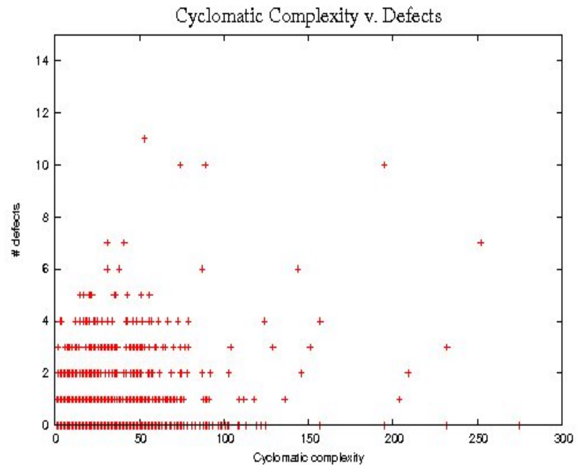
… defect
Quantification of effects by N-version (1994)



… defect
Relationship to static complexity ?

- There is little evidence that complexity measures such as the cyclomatic complexity v(G) are of any use at all in predicting defects

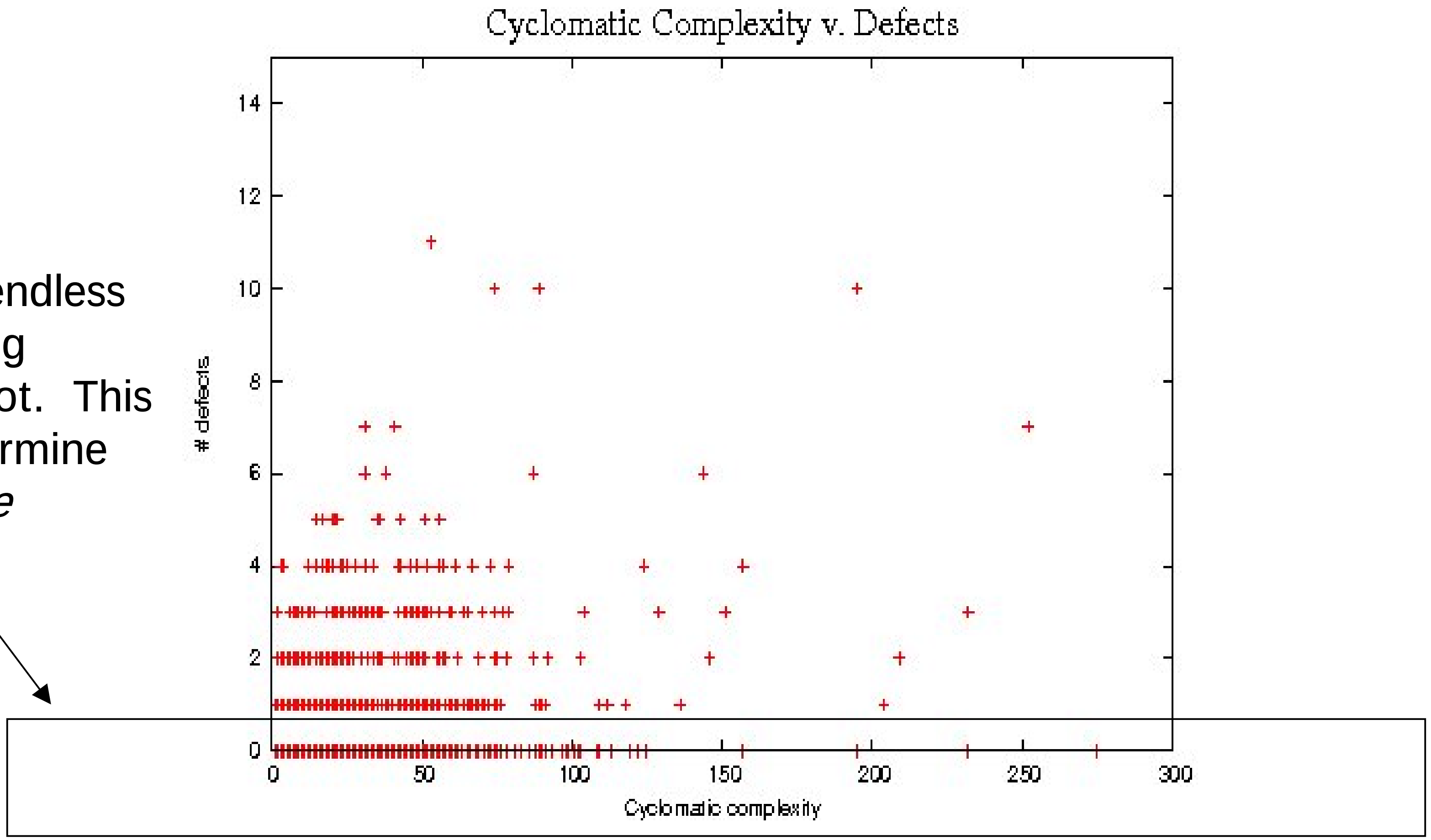
Defects



Cyclomatic number v(G)

NAG Fortran library over 25 years (Hopkins and Hatton (2008))

… defect
Is there anything unusual about ‘zero’ defect ?



PCA and endless rummaging suggest not. This may undermine *root-cause analysis*.

… programming language

- On static defects
  - Modern programming languages are littered with many types of statically detectable defect, (for example use of uninitialised variables).
  - These typically occur around 5-10 per 1000 lines of executable code and fail at an unacceptably high rate for high-integrity systems. They must be removed by tools plus inspections.
- Languages are astonishingly similar in their information properties …

… programming language

In any software system, conservation of size and information (i.e. choice) is overwhelmingly likely to produce a power-law alphabet distribution independently of programming language or application area. Hatton (2009).

$$p_i \sim (a_i)^{-\beta}$$

… programming languages

However for programming languages, a_i is made up of fixed and programmer-specified tokens

$$a_i = a_f + a_v(i)$$

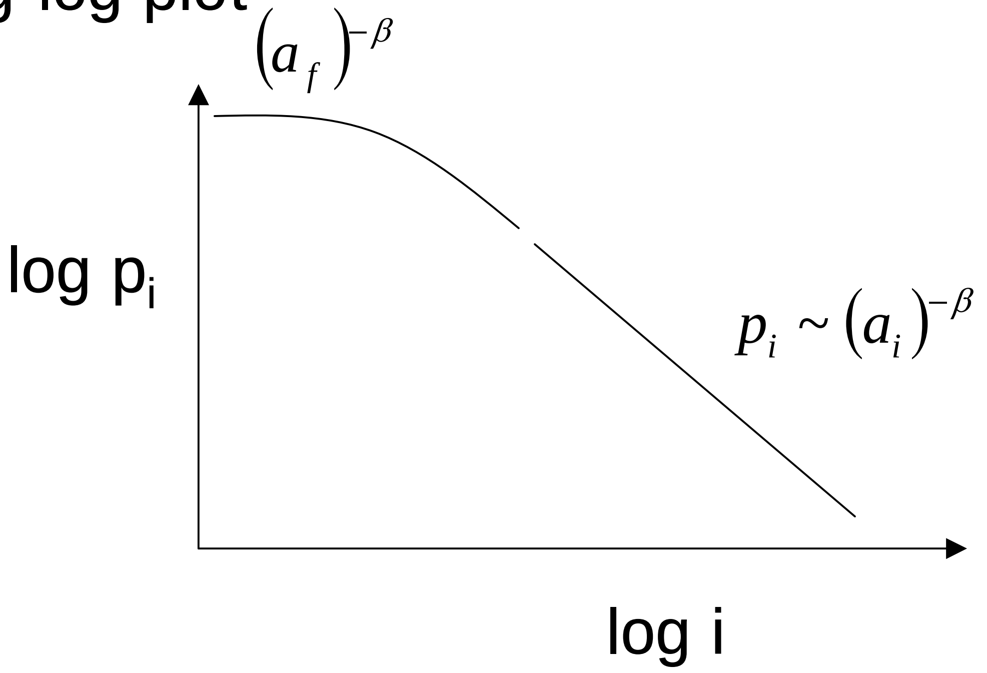
Fixed tokens of a language, **{ } [ ] ; while …**

Variable tokens, (id names and constants)

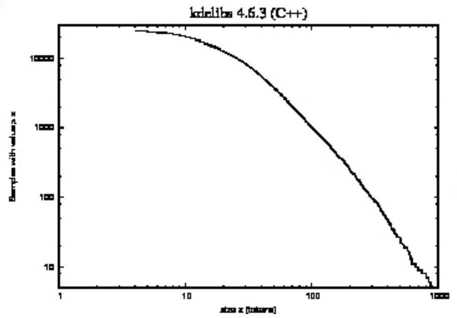
And for small components, a_i is dominated by a_f

… programming languages

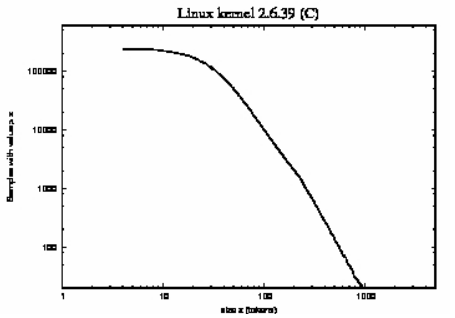
So we are looking for the following signature on a log-log plot



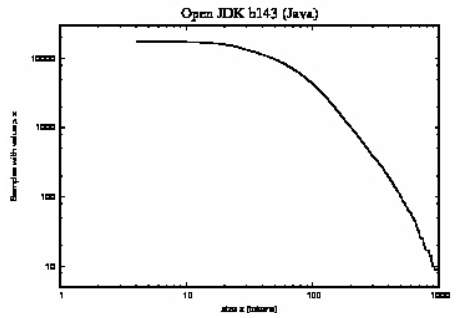
… programming languages Some results



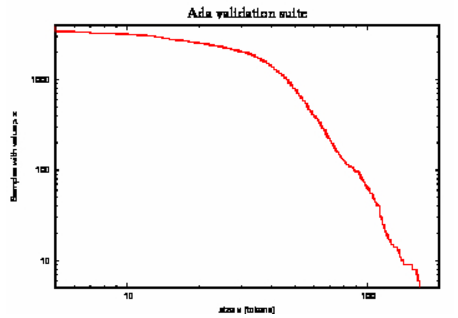
C++



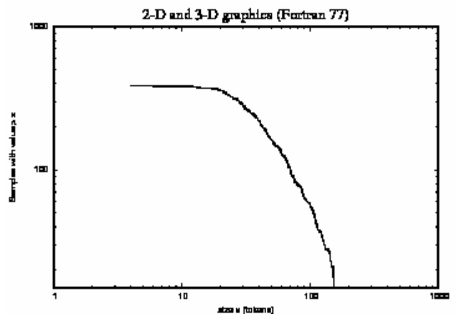
C



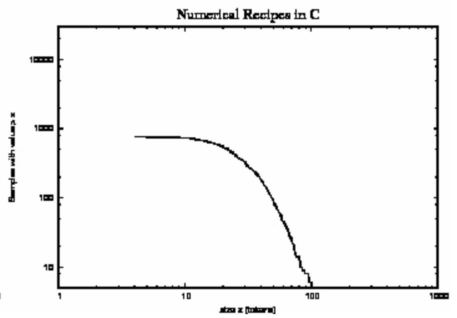
Java



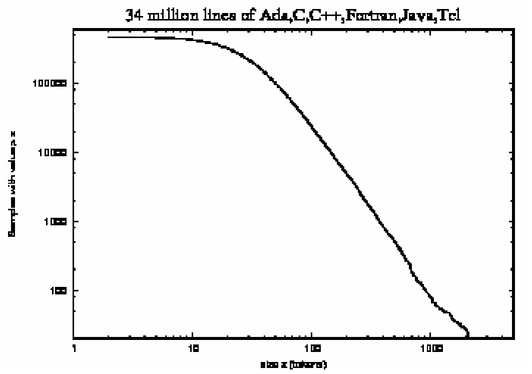
Ada



Fortran

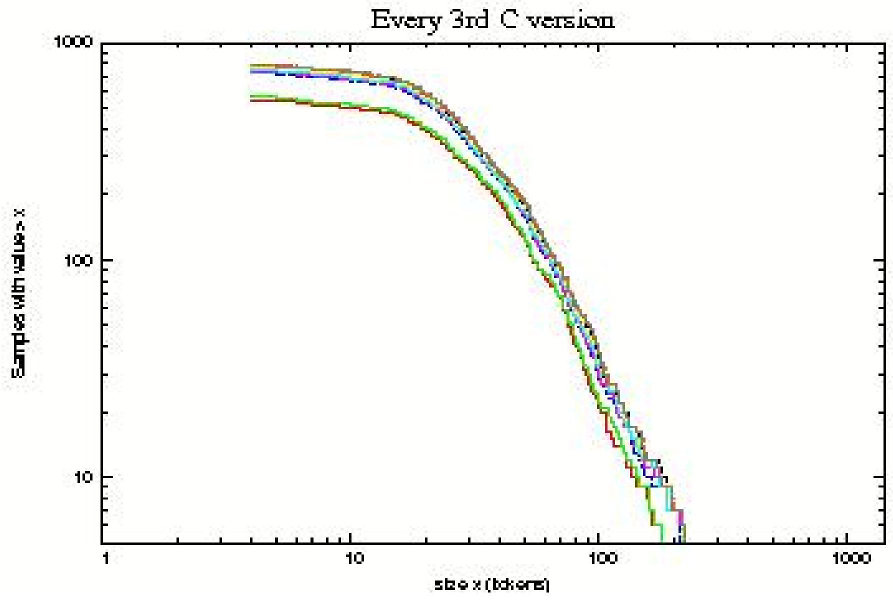


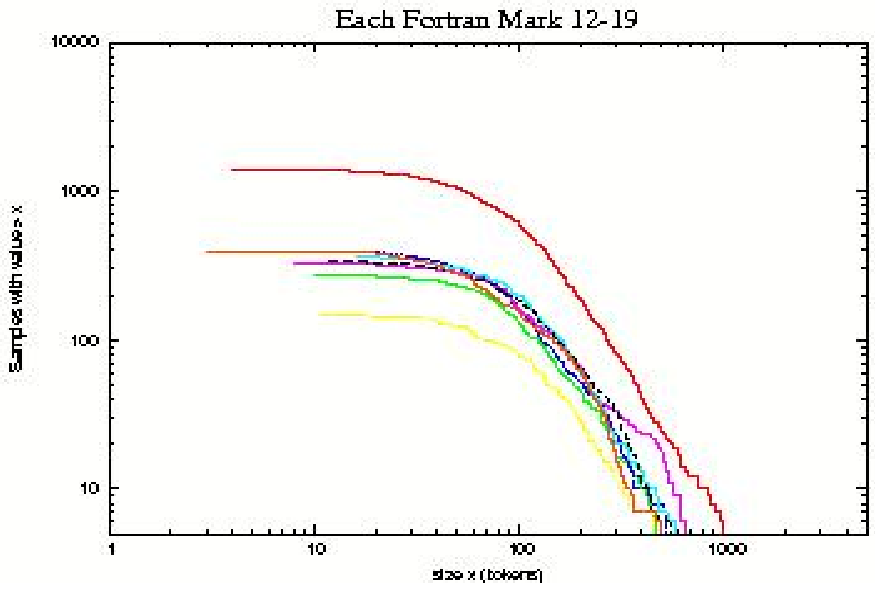
C Numerical



34 million lines of Ada, C, C++, Fortran, Java, Tcl in 75 systems.

… programming languages Is power-law behaviour persistent ?





… programming languages Defect clustering in the NAG Fortran library (over 25 years)

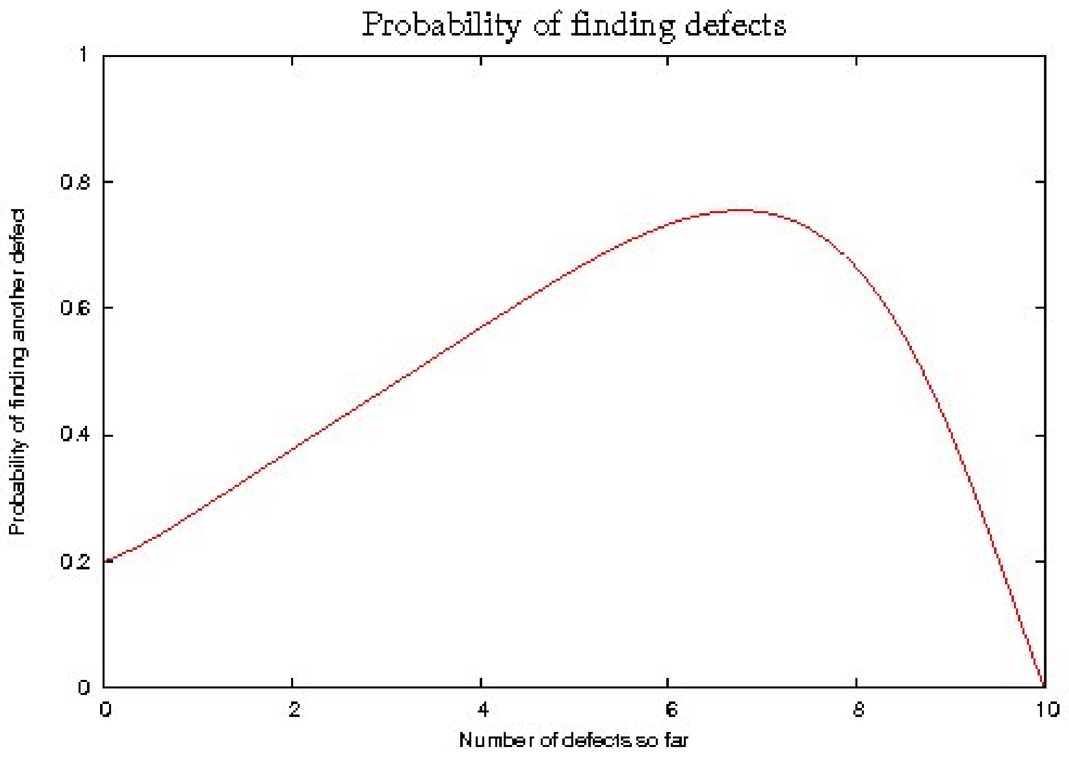
A simple model of defects leads to the prediction that defects will cluster

Zero-defect is like winning the lottery. There is no systematic way of achieving it.

Defects	components	XLOC
0	2865	179947
1	530	47669
2	129	14963
3	82	13220
4	31	5084
5	10	1195
6	4	1153
7	3	1025
> 7	5	1867

… programming languages

Clustering can be exploited: Conditional probability of finding defects *increases* initially*



* See, Hopkins and Hatton (2008), http://www.leshatton.org/NAG01_01-08.html

… process

- On process
  - Defined software processes including inspections, defect tracking and test planning are highly beneficial but beware of …
  - Box-ticking. Blind bureaucracy comes at a terrible price in safety-critical systems …

… process and box-ticking: Nimrod explosion, Afghanistan 2006

"The Nimrod Safety Case was a lamentable job from start to finish. It was riddled with errors, it missed key dangers, its production is a story of incompetence, complacency and cynicism."

Charles Haddon-Cave QC, Chairman of the enquiry on behalf of the House of Commons.

http://www.official-documents.gov.uk/document/hc0809/hc10/1025/1025.pdf

… FAQs and ways forward

- Some questions in High-Integrity development
  - Risk versus ingenuity
  - Knowledge transfer and Education
  - How is design approached ?
  - How are technological leaps tackled ?

High-Integrity Software

- High-Integrity Software and …
  - The scientific method
  - Defect
  - Programming languages
  - Process
  - FAQs and ways forward
- <u>Conclusions</u>

Conclusions

- High-Integrity software is as much about education as it is about technology
- Computing science is still struggling with defect quantification and damage limitation
- Scientific computation is not yet scientific
- For all kinds of reason, open source is of vital importance both for defect reduction and scientific reproducibility. There is great hope here and much to be excited about.

References

My writing site:-

http://www.leshatton.org/

Specifically,

http://www.leshatton.org/ICALEPSC_2011.html

Thanks for your attention.